

# ECHO Client Partner User Guide - Chapter 4

ECHO has been replaced by the **Common Metadata Repository (CMR)**, a high-performance, high-quality, continuously evolving metadata system that catalogs all data and service metadata records for the EOSDIS system and will be the authoritative management system for all EOSDIS metadata.

The information contained within this ECHO wiki is now archived for historical reference. Please navigate to the **CMR wiki pages**, or to the **CMR Overview page** on **Earthdata**.

- Querying for Earth Science Metadata
  - Formatting Your Query Results
  - Handling Large Result Sets
  - Visibility of Results
    - Restricted Items
    - Deleted Items
    - Querying for Orderable Data
  - Searching for Orbit Data
    - Backtrack orbit model
    - Orbit data representation
    - How ECHO Searches for Orbit Data
- Sample queries

## Guide Navigation

1. Preface
2. Before you begin
3. The basics
4. Logging in, setting up and getting started
5. **Querying for Earth Science metadata**
6. The ECHO Alternative Query Language
7. Ordering data through ECHO
8. Metadata subscriptions
9. Appendices

## Querying for Earth Science Metadata

ECHO uses the **ECHO Alternative Query Language (AQL)** for its querying capabilities. (See Chapter 6 for a detailed explanation of the language syntax.) To support potentially large queries, ECHO handles a query expression without regard to the details of whether and how it will return query results to the user.

An ECHO query consists of a query expression specified in XML, starting with <query>. All AQL queries must conform to the AQLQueryLanguage.dtd available on the ECHO website. Also, refer to Chapter 6, The ECHO Alternative Query Language for additional information about AQL.

You should validate your query against the DTD before passing it to ECHO. Since a query is passed to ECHO as a string, the query will not be validated against the DTD by the Web Service processing layer but rather at execution time, which, in the case of an asynchronous query or a subscription, could be after the Web Service call has completed.

You may use the optional argument **MaxResults** to limit the numbers of results returned from a query, which is useful when a query produces more data than can be processed. This also saves processing time for the query and presentation of results.

You may retrieve the results of a query in several different ways. The **ExecuteQuery** operation on the Catalog Service allows you to indicate whether you would like the data itself returned or simply the number of hits. When doing so, ensure that the **CatalogItemID** tag is returned. The string value under this tag is the actual catalog item GUID that you can use to order items from ECHO. You can specify this function in the **ResultType** argument with one of the following values:

**Table 2: Query Return Result Types**

Value	Description
RESULTS	Returns the detailed metadata for items that match the query directly in the response. When using this option, you may choose to limit the actual metadata values returned. In addition, ECHO will return a result set identifier (ResultSetGUID) for subsequent retrievals of the results or for paging through the result list. <i>Note that ECHO Operations limits the maximum number of items returned to 2,000 at a time.</i> The complete results are stored in your result set which you can retrieve by using the GetQueryResults operation.
RESULT_SET_GUID	Returns the result set guid of the results that are stored on the server. ECHO will generate a result set but will not return any results. You must subsequently retrieve the results using the GetQueryResults operation.
HITS	Returns the number of hits (matches) to the query and a ResultSetGUID for the results stored on the server. ECHO will generate a result set but will not return any results. The number of records may be a statistically determined for large result sets. You must subsequently retrieve the results using the GetQueryResults operation. Hits is a relatively expensive operation therefore if the client only needs to know if some data exists, it is faster to simply query for ITEM_GUIDS with a small iterator size.
ITEM_GUIDS	Returns the Catalog Item GUIDs that match the specified query. Note: No ResultSetGUID is returned since results do not persist in the system.  All the GUIDs of the granules/collections that satisfy the query are returned to the client. It is the client's responsibility to request the metadata for each individual granule/collection using the GetCatalogItemMetadata operation discussed later.

## Formatting Your Query Results

You can specify a subset of the information in the result set by using different parameters for the operations

**ExecuteQuery** and **GetQueryResults**.

The following elements are used to specify the format and content of a result set:

**Table 3: Result Set Content Elements**

Argument	Description
IteratorSize	Specifies the number of results to be returned from a single operation. This does not limit the number of items a query may match (see MaxResults) but limits to 2,000 the number of matching items returned in the result set, starting from the Cursor position.  This field is only used if the result type is set to RESULTS.
Cursor	Specifies the index of the first record to be returned in the result set. For example, a value of 5 will return results starting from the fifth record. If none is specified, it defaults to 1. If you repeat the same query later, use the same Cursor value.  This field is only used if the result type is set to RESULTS.
MetadataAttributes	Specifies which fields of the ECHO Metadata you actually want to return. By only requesting the parts of the metadata you are interested in, you can increase query performance substantially. By default, ECHO returns all of the metadata for each item.  This field is only used if the result type is set to RESULTS.

Metadata results are returned as XML that conforms to one of the following DTDs:

Granule metadata conforms to the Granule Results DTD—refer to Appendix F, Results DTDs (also located at: <http://api.echo.nasa.gov/echo/dtd/ECHOGranuleResults.dtd>).

Collection metadata conforms to the Collection Results DTD Appendix F, Results DTDs (also located at: <http://api.echo.nasa.gov/echo/dtd/ECHOCollectionResults.dtd>).

Metadata attributes are made up of two values: the XML metadata attribute name and a primitive type name. **ECHO currently ignores the type name.** The allowable metadata attribute names are specified in the appropriate DTD (ECHOGranuleResults.dtd for granules and ECHOCollectionResults.dtd for collections). If you specify a metadata attribute name that has sub-attributes, all of the sub-attributes will be included as well. For example, if you specify **Platform**, the following elements will be included in the metadata:

- Platform

- PlatformShortName
- Instrument
- InstrumentShortName
- Sensor
- SensorShortName
- SensorCharacteristics
- SensorCharacteristicName
- SensorCharacteristicValue

#### OperationMode

When you specify a sub-attribute, ECHO will return the parent attribute in the hierarchy as well as the sub-attribute. This allows you to ensure that data are correctly scoped. For example, if you specify **Sensor**, the following elements will be included in the metadata:

- Platform
- PlatformShortName
- Instrument
- InstrumentShortName
- GranuleUR
- GranuleURMetaData

***Detailed spatial attributes cannot be used as MetadataAttributes; only their containing element may be specified.*** For example, you cannot use **BoundingBox** as a MetadataAttribute, but you can use **HorizontalSpatialDomainContainer**. The following spatial elements cannot be specified as MetadataAttributes:

- Point
- Circle
- BoundingBox
- GPolygon
- Polygon
- PointLongitude
- PointLatitude
- CenterLongitude
- CenterLatitude
- Radius
- WestBoundingCoordinate
- NorthBoundingCoordinate
- EastBoundingCoordinate
- SouthBoundingCoordinate
- Boundary
- ExclusiveZone
- SinglePolygon
- MultiPolygon
- OutRing
- InnerRing

Specifying GranuleURMetaData as a MetadataAttribute is equivalent to not specifying any MetadataAttributes; the result set includes all the elements in the result DTD.

The following code snippet shows how to execute a query for all of the metadata for matching items.

### Code Listing 6: Executing a Simple Query

```
// Execute a query to get results
QueryResponse response = catalogService.executeQuery(userToken,
queryString, ResultType.RESULTS,
10, // Iterator
0, // Cursor
3000, // max results
null); // no metadata attributes specified

MetadataAttribute[] attributes = new MetadataAttribute[] { new
MetadataAttribute(
    "HorizontalSpatialDomainContainer", null) };

QueryResponse response = catalogService.executeQuery(userToken,
queryString, ResultType.RESULTS,
10, // Iterator
0, // Cursor
3000, // max results
attributes);
```

## Handling Large Result Sets

Given ECHO's large store of Earth Science data, it is possible for queries to return very large result sets. ECHO supports retrieving the results from a query in a number of ways. The simplest is to ask ECHO to return the results directly from the **ExecuteQuery** request by passing **RESULTS** as the **ResultType**. However, to prevent a single query from monopolizing ECHO resources, ECHO limits the number of results available in response to a query. By default, this limit is 2,000 items. ECHO Operations may change this limit depending on ECHO usage patterns.

For larger results, ECHO supports a paging mechanism. This allows you to page through the available data in page sizes that you select (up to the ECHO Operations configurable limit). For all **ResultTypes** ECHO will create and store a result set and return the corresponding GUID. You can page through the result set using the **GetQueryResults** operation. The arguments to **GetQueryResults** are similar to **ExecuteQuery** with the exception that you specify the result set GUID rather than a new query.

Result sets may change after they are created. Providers are continually changing the data they have registered in ECHO. New records may appear or may be removed from a result set. Because of this, you should watch the fields **Cursor** and **CursorAtEnd** when paging through a large result set:

**Cursor** specifies the index of the first record to be returned in the result set. For example, a value of 5 will return results starting from the fifth record. If none is specified, it defaults to 1. If you repeat the same query later, use the same **Cursor** value.

Use **CursorAtEnd** to determine when you have reached the end of the result set. This Boolean field is TRUE if the returned results were the last available results in the result set.

The following code illustrates paging through a result set and displaying it to the user.

### Code Listing 8: Paging Through Query Results

```
final int ITERATOR_SIZE = 10;
try
{
    CatalogServiceLocator catalogServiceLocator = new
CatalogServiceLocator();
    CatalogServicePort catalogService =
catalogServiceLocator.getCatalogServicePort();
    QueryResponse response = catalogService.executeQuery(userToken,
userQuery,
    ResultType.RESULT_SET_GUID, 0, 0, 1000, null);
    String resultSetGuid = response.getResults().getResultSetGuid();
    // begin paging through results
        int cursor = 1;
        boolean atEnd = false;
    while (!atEnd)
    {
        //Get next ITERATOR_SIZE results
        QueryResults results =
catalogService.getQueryResults(userToken,
        resultSetGuid, null, ITERATOR_SIZE, cursor);
        //Print out results
        System.out.println(results.getReturnData());
        //Set cursor to next index
        cursor = results.getCursor();
        //Check if at end of result set
        atEnd = results.isCursorAtEnd();
    }
    System.out.println("All results retrieved");
}
catch (EchoFault e)
{
    e.printStackTrace();
}
catch (ServiceException e)
{
    e.printStackTrace();
}
catch (RemoteException e)
{
    e.printStackTrace();
}
```

*Like ExecuteQuery, GetQueryResults takes an array of MetadataAttributes. Internally ECHO only stores in a result set the item IDs that match a given query. This means that you may pull different metadata from a single result set with each call by varying what you pass to the MetadataAttribute array without needing to re-query ECHO. It is highly recommended you use the MetadataAttribute array to restrict the information ECHO returns and thus improve performance.*

## Visibility of Results

When you execute a query, the query is applied to all the data in ECHO. However, when the results are retrieved, you may not see all of the items. What you can see depends on the rules defined by the Data Partners and the privileges granted to you.

## Restricted Items

If a particular item in your result set is restricted for you (i.e., you are not allowed to see it), based on your privileges, it will not be returned.

## Deleted Items

It is possible that between the time you execute a query and the time you view the results some of the matched items may have been deleted from ECHO or restricted due to a request from the Data Partner who owns the metadata. In that case, the item will not be returned in your result set. For more information about notification of deleted or restricted order items, refer to section 7.8.1, Restricted or Deleted Order Items.

## Querying for Orderable Data

Since ECHO 9.0, you have been able to exclude from your query data that cannot be ordered. Refer to section 1.1.1

## Searching for Orbit Data

### 4.4.1 Backtrack Orbit Search Algorithm

Orbit searching is by far the most accurate way to search for level 0-2 orbital swath data. Unfortunately orbital mechanics is a quite difficult field, and the most well known orbit model, the NORAD Propagator, is quite complex. The NORAD Propagator is designed to work with a wide range of possible orbits, from circular to extremely elliptical, and consequently requires quite a bit of information about the orbit to model it well.

To facilitate earth science, the orbits of satellites gathering earth science data are quite restricted compared to the variety of orbits the NORAD Propagator is designed to work with. Generally, the earth science community would like global coverage, with a constant field of view, at the same time every day. For this reason, most earth science satellites are in a sun-synchronous, near-polar orbit. Even missions that are not interested in global coverage, e.g., the Tropical Rainfall Measuring Mission (TRMM), are still interested in having a constant field of view so the coverage of the sensor is at a constant resolution. For this reason, ALL earth science satellites are in circular orbits. The Backtrack Orbit Search Algorithm, designed and developed by Ross Swick, exploits this fact to simplify the orbit model by modeling an orbit as a great circle under which the Earth rotates. This reduces the number of orbital elements required for the model from 22 to three. Moreover, the NORAD Propagator is designed to predict future orbits based on current status, and consequently must be reinitialized periodically to correct for cumulative error as the model spins forward. As the name implies Backtrack spins the orbit backwards, and in practice spins backwards at most one orbit, so there is no cumulative error.

For more information on Backtrack, please see <http://geospatialmethods.org/bosa/>.

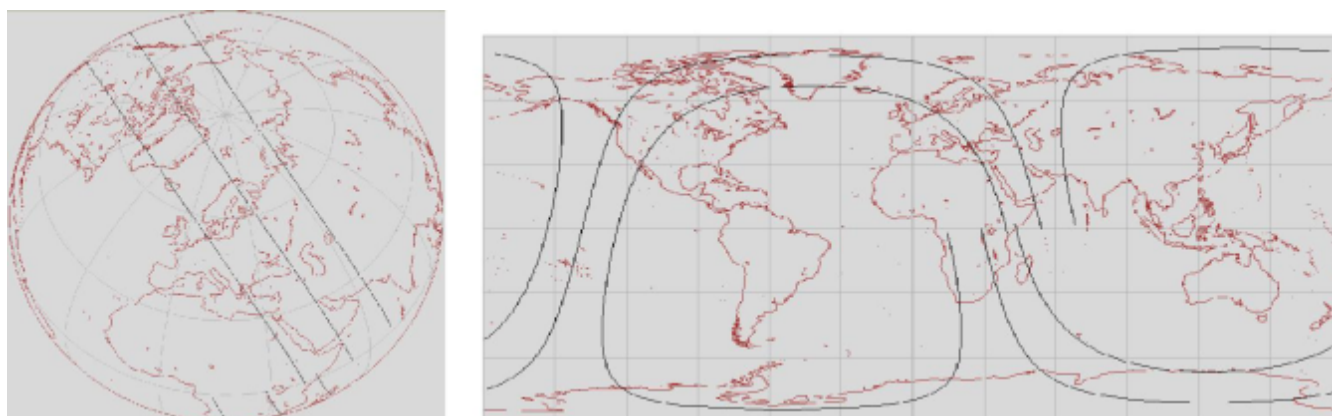
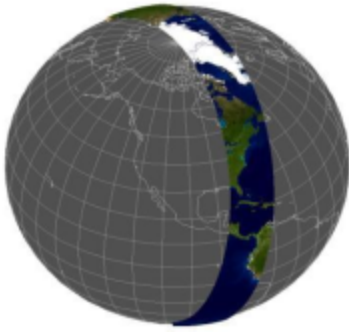


Figure 2. Typical Orbit Path Represented on a Globe and the same Path on a Map

## Backtrack orbit model



Three parameters to define an orbit:

1. Instrument swath width (in kilometers)
2. Satellite declination or inclination (in degrees)
3. Satellite period (in minutes)

## Orbit data representation

Three parameters to represent orbit data:

1. Equatorial crossing longitude
2. Start circular latitude (or start latitude and start direction)
3. End circular latitude (or end latitude and end direction)

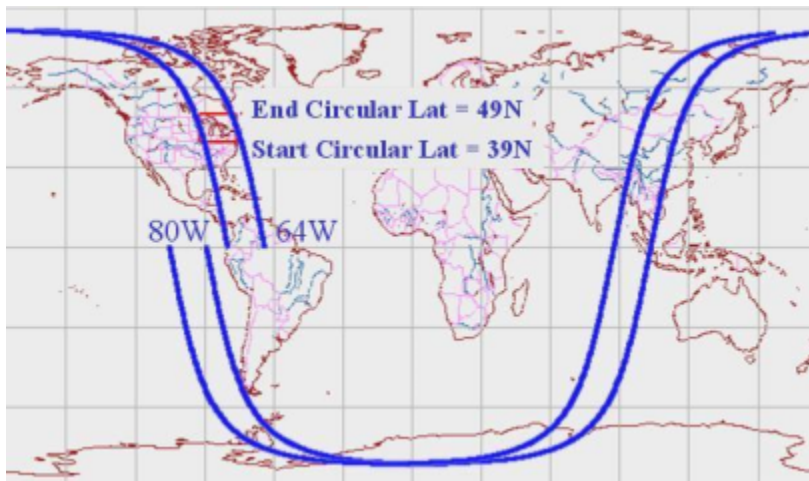
## How ECHO Searches for Orbit Data

- The user specifies a regular spatial window

**Figure 3. Spatial Window**

```
<granuleCondition>
  <spatial>
    <IIMSPolygon>
      <IIMSLRing>
        <IIMSPoint lon="-90" lat="49" />
        <IIMSPoint lon="-90" lat="39" />
        <IIMSPoint lon="-70" lat="39" />
        <IIMSPoint lon="-70" lat="49" />
        <IIMSPoint lon="-90" lat="49" />
      </IIMSLRing>
    </IIMSPolygon>
    <SpatialType>
      <list>
        <value>ORBIT</value>
      </list>
    </SpatialType>
  </spatial>
</granuleCondition>
```

- Backtrack then calculates from both ascending and descending a path for equatorial longitude crossings and start/end circular latitudes according to user's query window



## Sample queries

The following are sample queries that you can execute against ECHO. Note that the provider and the datasets used in these samples are representative only; you should modify the query to suit your needs.

### Code Listing 9: Sample Collection Query (Discovery Search)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query PUBLIC "-//ECHO CatalogService (v{*}10{*})//EN"
"http://api.echo.nasa.gov/echo/dtd/IIMSAQLQueryLanguage.dtd">
<!-- Search for collections from ORNL_DAAC that have parameter value that
contains 'IMAGERY'-->
<query>
  <for value="collections"/>
  <dataCenterId>
    <list>
      <value>ORNL_DAAC</value>
    </list>
  </dataCenterId>
  <where>
    <collectionCondition>
      <parameter>
        <textPattern>%Imagery%</textPattern>
      </parameter>
    </collectionCondition>
  </where>
</query>
```

### Code Listing 10: Sample Collection Query from Two Providers (Discovery Search)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query PUBLIC "-//ECHO CatalogService (v{*}10{*})//EN"
"http://api.echo.nasa.gov/echo/dtd/IIMSAQLQueryLanguage.dtd">
<!-- Search for collections from GSFCECS and ORNL_DAAC that have
processing level 1A or 2 -->
<query>
  <for value="collections"/>
  <dataCenterId>
    <list>
      <value>GSFCECS</value>
      <value>ORNL_DAAC</value>
    </list>
  </dataCenterId>
  <where>
    <collectionCondition negated="y">
      <processingLevel>
        <list>
          <value>'1A'</value>
          <value>'2'</value>
        </list>
      </processingLevel>
    </collectionCondition>
  </where>
</query>
```

### Code Listing 11: Sample Collection Query with Temporal and Spatial Constraints

(Discovery)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query PUBLIC "-//ECHO CatalogService (v{*}10{*})//EN"
"http://api.echo.nasa.gov/echo/dtd/IIMSAQLQueryLanguage.dtd">
<!-- Search for collections from ORNL_DAAC with: temporal range:
periodic range between Jan 1, 1990 and Dec. 31 1998from the 1st to the
300th day of each year, AND spatial extent: bounding box 60S, 70W to
60N, 70E. -->
<query>
  <for value="collections"/>
  <dataCenterId>
    <value>ORNL_DAAC</value>
  </dataCenterId>
  <where>
    <collectionCondition>
      <temporal>
        <startDate>
          <Date YYYY="1990" MM="01" DD="01"/>
        </startDate>
        <stopDate>
          <Date YYYY="1998" MM="12" DD="31"/>
        </stopDate>
        <startDay value="1"/>
        <endDay value="300"/>
      </temporal>
    </collectionCondition>
    <collectionCondition negated="n">
      <spatial operator="RELATE">
        <IIMSPolygon>
          <IIMSLRing>
            <IIMSPoint long='-10' lat='85'/> <IIMSPoint long='10' lat='85'/>
            <IIMSPoint long='10' lat='89'/> <IIMSPoint long='-10' lat='89'/>
            <IIMSPoint long='-10' lat='85'/>
          </IIMSLRing>
        </IIMSPolygon>
      </spatial>
    </collectionCondition>
  </where>
</query>
```

### Code Listing 12: Sample Collection Query with Complex Temporal and Spatial Conditions

(Discovery)

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query PUBLIC "-//ECHO CatalogService (v{*}10{*})//EN"
"http://api.echo.nasa.gov/echo/dtd/IIMSAQLQueryLanguage.dtd">
<!-- Search for collections from ORNL_DAAC with
temporal range: periodic range between Jan 1, 1990 and Dec. 31 1998 from
the 1st to the 300th day of each year, AND
some days of January. source name: L7 or AM-1 AND
spatially covering any 'temperate' region or USA -->
<query>
  <for value="collections"/>
  <dataCenterId>
    <list>
      <value>ORNL/value>
    </list>
  </dataCenterId>
  <where>
    <collectionCondition>
      <temporal>
        <startDate>
          <Date YYYY="1990" MM="01" DD="01"/>
        </startDate>
        <stopDate>
          <Date YYYY="1998" MM="12" DD="31"/>
        </stopDate>
        <startDay value="1"/>
        <endDay value="300"/>
      </temporal>
    </collectionCondition>
    <collectionCondition negated='n'>
      <sourceName>
        <list>
          <value>'L7'</value>
          <value>'AM-1'</value>
        </list>
      </sourceName>
    </collectionCondition>
    <collectionCondition>
      <spatialKeywords>
        <list>
          <value>'temperate'</value>
          <value>'USA'</value>
        </list>
      </spatialKeywords>
    </collectionCondition>
    <collectionCondition>
      <temporalKeywords>
        <textPattern>'%january%'</textPattern>
      </temporalKeywords>
    </collectionCondition>
  </where>
</query>

```

### Code Listing 13: Sample Collection Query Using Provider Specific Attributes (Discovery)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query PUBLIC "-//ECHO CatalogService (v10)//EN"
"http://api.echo.nasa.gov/echo/dtd/IIMSAQLQueryLanguage.dtd">
<query>
  <for value="collections"/>
  <dataCenterId>
    <value>ORNL_DAAC</value>
  </dataCenterId>
  <where>
    <collectionCondition>
      <additionalAttributeNames>
        <list>
          <value>'Provider_Specific_Attribute_1'</value>
          <value>'Provider_Specific_Attribute_3'</value>
        </list>
      </ additionalAttributeNames >
    </collectionCondition>
  </where>
</query>
```